TDDE68 + TDDE47

Lecture 3: Processes, Threads and File Systems (I)

Klas Arvidsson

Based on slides by Mikael Asplund and Adrian Pop

Thanks to Christoph Kessler and Simin Nadjm-Tehrani for some of the material behind these slides



Today's lecture

- Processes
 - Concepts
 - Creation, switching and termination
- Threads
- Process interaction
 - 2 slide introduction
- File Systems (I)
 - Introduction

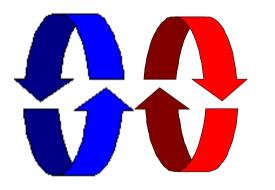
Reading guidelines

- Silberschatz et al. (10th ed.)
 - Chapter 3.1-3.4, 4.1-4.3, 4.5, 13.1, 14.1-2

Concurrent Programs



A **sequential** program has a single thread of control.



A **concurrent** program has multiple threads of control allowing it perform multiple computations "in parallel" and to control multiple external activities which occur at the same time.



[Magee and Kramer 2006]

Related terms

 Concurrent programs 	 Define actions that may be performed simultaneously
 Parallel programs 	 A concurrent program that is designed for execution on parallel hardware
 Distributed programs 	 Parallel programs designed to run on network of autonomous processors that do not share memory



Concurrency on a single core CPU

• Concurrency can be achieved through preemptive multitasking

 Let each program run for a short while and then switch to the next one

 Improvement over the "multiprogramming" paradigm



The abstract notion of **Process**

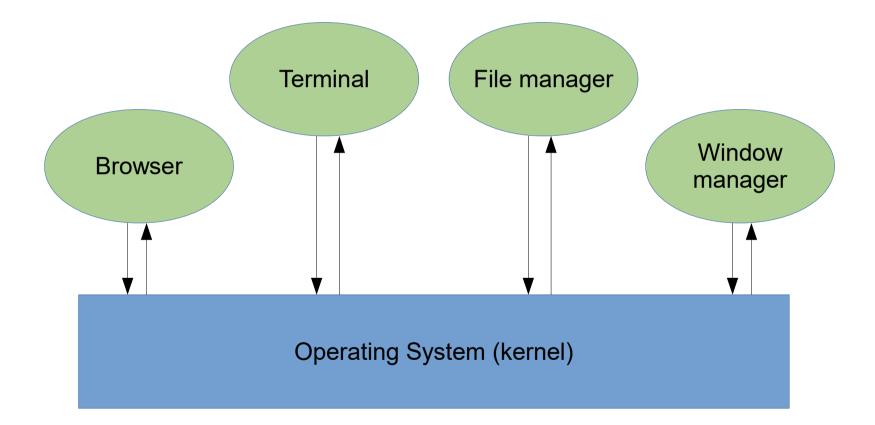
- An abstraction in computer science used for describing program execution and potential parallelism
- What other abstractions do you know?
 - Functions
 - Classes, Objects, Methods
- Processes emphasise the run-time behaviour



Typical OS terminology:

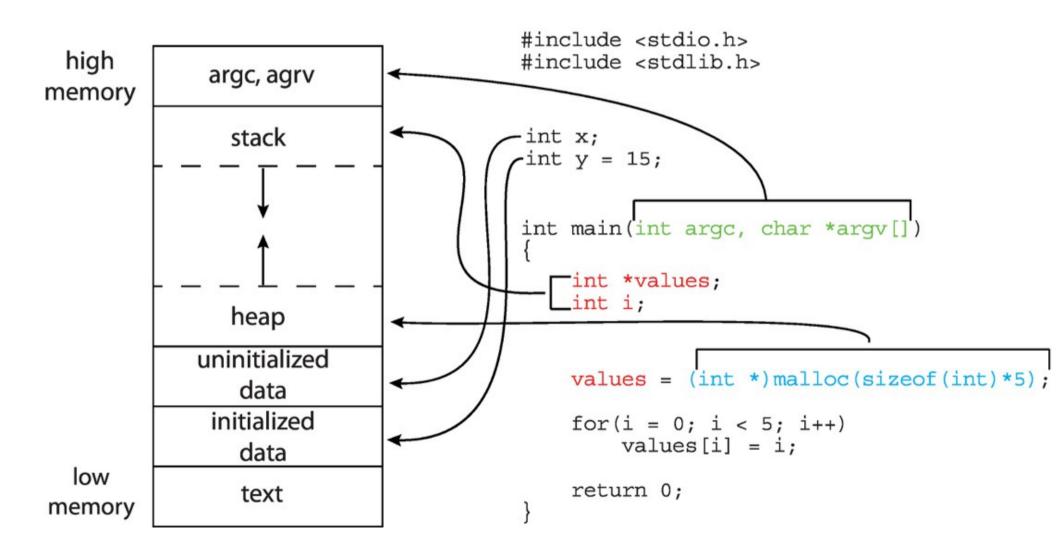
A process is a program in execution with its own memory

Example processes



Process management is one of the key tasks of an operating system.

Process in Memory



Process Control Block (PCB)

Information associated with each process

- Process Identifier (PID)
- Process state
- CPU registers
- Program counter
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process representation in Linux

https://github.com/torvalds/linux/blob/master/include/linux/sched.h

C 0	linux/sche	d.h at master · torvalds × +			~			×
<pre># define TASK_NUMING # 0x0000 # madrine TASK_INTERRUPTIELE # 0x0001 # madrine TASK_INTERRUPTIELE # 0x0002 # madrine TASK_INTERRUPTIELE # 0x0002 # madrine TASK_INTERRUPTIELE # 0x0008 # /* Used in tsk-posit_state: */ # madrine EXIT_DOME # 0x0008 # /* Used in tsk-posit_state: */ # madrine TASK_DOME # 0x0008 # madrine TASK_STATE_MAX # 0x2000 #################################</pre>	$\leftrightarrow \rightarrow c$	☆	aster/include/linux/sched.h		•	0 1	-	• :
# define TASK_INTERRUPTIBLE 0x0001 # define TASK_INTERRUPTIBLE 0x0002 # defineTASK_INPED 0x0008 # defineTASK_INPED 0x0008 # define EXI_DEAD 0x0008 # define EXI_TZWBIE 0x0001 # define EXI_TZWBIE 0x0020 # define EXI_TZWBIE 0x0020 # define EXI_TACKE (EXIT_ZOWIE EXIT_DEAD) # define TASK_NARED 0x0040 # define TASK_NARED 0x0040 # define TASK_NARE 0x00400 # define TASK_NARE 0x02000 # define TASK_NARE 0x10400 # define TASK_NARE 0x20400 # define TASK_NARE 0x20400 # de			8y2000					^
#define TAS_UNINTERRUPTIBLE 0x0002 #define TAS_STOPPD 0x0003 #define TAS_STOPPD 0x0003 #define EXT_DEAD 0x00030 #define EXT_DEAD 0x00230 #define EXT_DEAD 0x00230 #define EXT_DEAD 0x00230 #define EXT_DEAD 0x00230 #define EXT_DEAD 0x00240 #define TAS_PARKED 0x00240 #define TAS_PARKED 0x0040 #define TAS_NAKED 0x0040 #define TAS_NAKED 0x0040 #define TAS_NAKENL 0x01000 #define TAS_NAKENL 0x02000 #define TAS_STOPED (TAS_CAUNTATION \$10000 #define TAS_STOPED (TAS_CAUNTERUPTIBLE) #define TAS_STOPED (TAS_CAUNTERUPTIBLE] TAS_NONDAD #define TAS_NONDAL (TAS_AUNTERUPTIBLE] TAS_NONDAD #define TAS_STOPED (TAS_AUNTERUPTIBL								- 1
#defineTAS_TACED 0x0004 # defineTAS_TACED 0x0008 # defineTAS_TACED 0x0010 # define EXIT_DEAD 0x0010 # define EXIT_TACE (EXIT_ZOWBIE EXIT_DEAD) # define EXIT_TACE (EXIT_ZOWBIE EXIT_DEAD) # define EXIT_TACE (EXIT_ZOWBIE EXIT_DEAD) # define TASK_MARED 0x00010 # define TASK_MARED 0x00010 # define TASK_MARED 0x00010 # define TASK_MARED 0x00010 # define TASK_MARED 0x00000 # define TASK_MARED 0x00000 # define TASK_MARED 0x00000 # define TASK_MAREM 0x02000								
#define _TASK_TRACED 0x0008 /* Used in tsk:>sott_tate: */ #define EXIT_TOAD 0x0010 #define EXIT_TOAD 0x0010 #define EXIT_TOAD 0x0010 #define EXIT_TOAD 0x0010 #define EXIT_TOACE (EXIT_ZOMBIE EXIT_TOAD) /* Used in tsk:>state again: */ #define TASK_TOAD 0x0000 #define TASK_DRAFED 0x0010 #define TASK_NARKELL 0x0100 #define TASK_NARKELL 0x0200 #define TASK_NARKELL 0x0200 #define TASK_NOD 0x0200 #define TASK_NOD 0x0200 #define TASK_NON 0x0200 #define TASK_NON 0x0200 #define task_NOLOAD 0x0200 #define task_NOLOAD 0x0200 #define task_NOLOAD 0x0200 #define task_NOLOAD 0x0200 #define task_STARCED (TASK_MAREKILL _TASK_STOPED) #define <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th>								
<pre>8 /* Used in tsk-sexit_state: */ 8 #define EXT_DEAD</pre>								
#define EXIT_DEAD 0x0020 #define EXIT_ZOWBIE 0x0020 #define EXIT_TAKE (EXIT_ZOWBIE EXIT_DEAD) /* Used in tsk->state again: */ #define TAK_PARED 0x0040 #define TAK_GEAD 0x00800 #define TAK_MOLAD 0x00800 #define TAK_MOLAD 0x0000 #define TAK_MELOCL_WAIT 0x0000 #define TAK_TALCLAT 0x1000 #define TAK_STAFE_MAX 0x20000 #define TAK_STAFE_MAX 0x2000 #define TAK_STAFE_D (TAK_MAKEKILL _TAK_MINHERUPTIBLE) #define TAK_TOLE (TAK_MAKEKILL _TAK_MOLAD) #define TAK_MORAL (TAK_MAKEKILL _TAK_MOLAD) #define TAK_MORAL (TAK_UNINTERRUPTIBLE TAK_UN								
99 #define EXIT_ZONBIE 0x0020 91 #define EXIT_TRACE (CXIT_ZONBIE EXIT_DEAD) 92 /* Used in tak-systate again: */ 91 #define TASK_PARKED 0x0040 92 #define TASK_DEAD 0x0040 93 #define TASK_DEAD 0x00100 94 #define TASK_MAKENI 0x0100 95 #define TASK_MAKENI 0x0100 96 #define TASK_MAKENI 0x0100 97 #define TASK_MAKENI 0x0100 98 #define TASK_NOLAD 0x0400 99 /* R1 specific auxilliary flag to mark R1 lock waiters */ mdefine TASK_STATE_MAX 0x2000 90 /* Convenience macros for the sake of set_current_state: */ mdefine TASK_STATE_MAX 0x2000 91 #define TASK_STATE_MAX 0x2000 100 92 /* Convenience macros for the sake of set_current_state: */ 113 mdefine TASK_STATE_MAX 0x2000 93 /* Convenience macros for the sake of set_current_state: */ 114 mdefine TASK_STATE_MAX 0x2000 94 #define TASK_STATE_MAX 0x2000 116 115 116			0x0010					
92 /* Used in tsk->state again: */ 93 #define TASK_PARED 0x0080 94 #define TASK_MARKELL 0x0080 95 #define TASK_MARKELL 0x0100 96 #define TASK_MARKELL 0x0200 97 #define TASK_NAKKELL 0x0200 98 #define TASK_NAKKELL 0x0200 99 /* RT specific auxilliary flag to mark RT lock waiters */ 90 #define TASK_NEW 0x2000 91 #define TASK_KTLOK_WAIT 0x1000 92 /* Convenience macros for the sake of set_current_state: */ #define TASK_KTLLABLE 93 /* Convenience macros for the sake of set_current_state: */ #define TASK_STOPPED (TASK_MAKEKILL _TASK_STOPPED) 94 #define TASK_TALEE (TASK_MAKEKILL _TASK_STOPPED) #define TASK_TARCED 95 #define TASK_TALEE (TASK_MAKEKILL _TASK_NOLOAD) #define TASK_TARCED 96 #define TASK_NORMAL (TASK_MAKEKILL _TASK_NOLOAD) #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNITERRUPTIBLE) 97 #define TASK_NORMAL (TASK_MAKERUPTIBLE TASK_UNITERRUPTIBLE \ #define TASK_REPORT TASK_TARCED EXIT_DEAD EXIT_DOMBIE \			0x0020					
93 #define TASK_PARKED 0x0040 94 #define TASK_DEAD 0x0080 95 #define TASK_MAKEKILL 0x0100 96 #define TASK_WAKEKILL 0x0100 97 #define TASK_WAKEKILL 0x0100 98 #define TASK_WAKEKIL 0x0200 97 #define TASK_WAKEKIL 0x0200 98 #define TASK_NDAD 0x0400 99 /* RT specific auxiliary flag to mark RT lock waiters */ 0x0200 98 #define TASK_STATE_MAX 0x2000 99 /* Convenience macros for the sake of set_current_state: */ 100 90 #define TASK_STOPED (TASK_MAKEKILL _TASK_STOPED) 91 #define TASK_STOPED (TASK_MAKEKILL _TASK_STOPED) 92 #define TASK_IDLE (TASK_MAKEKILL _TASK_NOLOD) 93 #define TASK_IDLE (TASK_MAKEKILL _TASK_NOLOD) 94 #define TASK_STACED (TASK_MAKEKILL _TASK_NOLOD) 95 #define TASK_IDLE (TASK_MAKEKILL _TASK_NOLOD) 96 #define TASK_NOMAL (TASK_UNINTERRUPTIBLE TASK_NOLOD) 97 #define TASK_NOMAL (TASK_UNINTERRUPTIBLE TASK		#define EXIT_TRACE	(EXIT_ZOMBIE EXIT_DEAD)					
94 #define TASK_WAKERUL 9x0880 95 #define TASK_WAKERUL 9x0800 96 #define TASK_WAKING 9x0800 97 #define TASK_NOLOD 9x0480 98 #define TASK_NOLOD 9x0480 99 /* RT specific auxilliary flag to mark RT lock waiters */ #define TASK_RILOK_MAIT 9x0800 90 /* Convenience macros for the sake of set_current_state: */ #define TASK_STOPED (TASK_MAKEKILL _TASK_UNINTERRUPTIBLE) 90 /* Convenience macros for the sake of set_current_state: */ #define TASK_STOPED (TASK_MAKEKILL _TASK_UNINTERRUPTIBLE) 90 #define TASK_STOPED (TASK_MAKEKILL _TASK_STOPED) #define TASK_STOPED 91 #define TASK_TARCED (TASK_MAKEKILL _TASK_TRACED) #define TASK_TOLE 91 #define TASK_NORMAL (TASK_MAKEKILL _TASK_UNINTERRUPTIBLE) #define TASK_NORMAL (TASK_UNINTERRUPTIBLE TASK_UNINTERRUPTIBLE) 91 #define TASK_NORMAL (TASK_KUNING TASK_INTERRUPTIBLE \ #define TASK_REPORT (TASK_UNINTERRUPTIBLE \ 91 /* get_task_state(): */ #define TASK_REPORT (TASK_UNINTERRUPTIBLE _TASK_STOPPED \		/* Used in tsk->state again: */						
95 #define TASK_MAKEKILL 0x8100 96 #define TASK_MAKEKILL 0x8200 97 #define TASK_MAKEKING 0x8300 97 #define TASK_NOLOAD 0x8800 99 /* RT specific auxilliary flag to mark RT lock waiters */ #define TASK_STATE_MAX 0x2000 101 #define TASK_STATE_MAX 0x2000 #define TASK_STATE_MAX 0x2000 102 /* Convenience macros for the sake of set_current_state: */ #define TASK_KILABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 103 /* Convenience macros for the sake of set_current_state: */		#define TASK_PARKED	0x0040					
96 #define TASK_NAKING 0x0200 97 #define TASK_NEW 0x0800 98 #define TASK_NEW 0x0800 99 /* RT specific auxilliary flag to mark RT lock waiters */ #define TASK_RTLOCL_MAIT 0x1000 101 #define TASK_STATE_MAX 0x2000 #define TASK_STATE_MAX 0x2000 102 /* Convenience macros for the sake of set_current_state: */ #define TASK_KTLLABLE (TASK_MAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_STOPPED (TASK_MAKEKILL _TASK_STOPPED) #define TASK_TRACED 106 #define TASK_TRACED (TASK_WAKEKILL _TASK_TRACED) #define TASK_TRACED 107 #define TASK_NORMAL (TASK_UNINTERRUPTIBLE TASK_NOLOAD) #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 118 /* get_task_state(): */ #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ _TASK_TRACED EXIT_ZOMBIE \ _TASK_TRACED EXIT_ZOMBIE \ _TASK_PARKED) #define \		#define TASK_DEAD	0x0080					
97 #define TASK_NOLOAD ex0400 98 #define TASK_NEW 0x0800 97 *R T specific auxilliary flag to mark RT lock waiters */ 108 #define TASK_RTLOCK_MAIT 0x1000 109 #define TASK_STATE_MAX 0x2000 101 #define TASK_STATE_MAX 0x2000 102 /* Convenience macros for the sake of set_current_state: */ 104 #define TASK_KILLABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_TRACED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL TASK_TRACED) 102 #define TASK_TOPED (TASK_WAKEKILL TASK_TRACED) 108 #define TASK_TOLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 109 #define TASK_MORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 110 /* get_task_state(): */ #define TASK_REPORT (TASK_UNINTERRUPTIBLE TASK_STOPPED \ 113 /* get_task_state(): */ #define TASK_REPORT (TASK_UNINTERRUPTIBLE TASK_STOPPED \ 116 TASK_TRACED EXIT_DEAD EXIT_ZOMBIE \ TASK_PARKED) 118		#define TASK_WAKEKILL	0x0100					
98 #define TASK_NEW exx0800 99 /* RT specific auxilliary flag to mark RT lock waiters */ 108 #define TASK_RTLOCK_MAIT exx1000 109 #define TASK_STATE_MAX exx2000 101 #define TASK_STATE_MAX exx2000 102 /* Convenience macros for the sake of set_current_state: */ 104 104 #define TASK_STATE_MAX exx2000 105 #define TASK_STOPPED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL _TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL _TASK_TRACED) 107 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_MOLOAD) 108 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 109 /* Convenience macros for the sake of wake_up(): */ #define TASK_NORMAL 117 #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ 118		<pre>#define TASK_WAKING</pre>	0x0200					
<pre>99 /* RT specific auxilliary flag to mark RT lock waiters */ 109 #define TASK_RTLOCK_WAIT 9x1000 101 #define TASK_STATE_MAX 9x2000 102 103 104 #define TASK_STATE_MAX 9x2000 105 105 106 #define TASK_KILLABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_KILLABLE (TASK_WAKEKILL _TASK_UNINTERRUPTIBLE) 106 #define TASK_STOPPED (TASK_WAKEKILL _TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL _TASK_TRACED) 107 108 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 109 109 109 109 109 109 109 109 109 109</pre>		#define TASK_NOLOAD	0x0400					
100 #define TASK_RTLOCK_WAIT 0x1000 101 #define TASK_STATE_MAX 0x2000 102 /* Convenience macros for the sake of set_current_state: */ 104 #define TASK_KILLABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_STOPPED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL TASK_TRACED) 107 */* Convenience macros for the sake of wake_up(): */ 108 #define TASK_TOPE 109 /* Convenience macros for the sake of wake_up(): */ 101 /* Convenience macros for the sake of wake_up(): */ 102 /* get_task_state(): */ 103 /* get_task_state(): */ 104 #define TASK_REPORT 105 TASK_UNINTERRUPTIBLE TASK_STOPPED \ 106		#define TASK_NEW	0x0800					
101 #define TASK_STATE_WAX 0x2000 102 /* Convenience macros for the sake of set_current_state: */ 104 #define TASK_KILLABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_STOPPED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TARCED (TASK_WAKEKILL TASK_TRACED) 107 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 108 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 110 /* Convenience macros for the sake of wake_up(): */ 111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112 /* get_task_state(): */ 113 /* get_task_state(): */ 114 #define TASK_REPORT (TASK_UNNING TASK_STOPPED \ 115 TASK_UNINTERRUPTIBLE TASK_STOPPED \ 116 TASK_TRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED)		/* RT specific auxilliary flag to mark	RT lock waiters */					
102 103 /* Convenience macros for the sake of set_current_state: */ 104 #define TASK_KILLABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_STOPPED (TASK_WAKEKILL _TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL _TASK_TRACED) 107		<pre>#define TASK_RTLOCK_WAIT</pre>	0x1000					
103 /* Convenience macros for the sake of set_current_state: */ 104 #define TASK_KILLABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_STOPPED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL TASK_TRACED) 107 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 109 /* Convenience macros for the sake of wake_up(): */ 111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112 /* get_task_state(): */ 113 /* get_task_state(): */ 114 #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ 115 TASK_UNINTERRUPTIBLE TASK_STOPPED \ 116 TASK_TRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED)		<pre>#define TASK_STATE_MAX</pre>	0x2000					
104 #define TASK_KILLABLE (TASK_WAKEKILL TASK_UNINTERRUPTIBLE) 105 #define TASK_STOPPED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL TASK_TRACED) 107 108 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 109 /* Convenience macros for the sake of wake_up(): */ 111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112 /* get_task_state(): */ 113 /* get_task_state(): */ 114 113 /* get_task_state(): */ TASK_UNINTERRUPTIBLE TASK_STOPPED \ 115 114 #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ 115 115 TASK_UNINTERRUPTIBLE TASK_STOPPED \ 116 TASK_TRACED EXIT_ZOMBIE \ 117 TASK_PARKED) 118 118 118								
105 #define TASK_STOPPED (TASK_WAKEKILL TASK_STOPPED) 106 #define TASK_TRACED (TASK_WAKEKILL TASK_TRACED) 107 108 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 109 100		/* Convenience macros for the sake of s	et_current_state: */					
106 #define TASK_TRACED (TASK_WAKEKILL TASK_TRACED) 107		#define TASK_KILLABLE	(TASK_WAKEKILL TASK_UNINTERRUPTIBLE)					
107 108 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 109 110 /* Convenience macros for the sake of wake_up(): */ 111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112		#define TASK_STOPPED	(TASK_WAKEKILL TASK_STOPPED)					
108 #define TASK_IDLE (TASK_UNINTERRUPTIBLE TASK_NOLOAD) 109 /* Convenience macros for the sake of wake_up(): */ 110 /* Convenience macros for the sake of wake_up(): */ 111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112		#define TASK_TRACED	(TASK_WAKEKILL TASK_TRACED)					
109 110 /* Convenience macros for the sake of wake_up(): */ 111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112								
110 /* Convenience macros for the sake of wake_up(): */ 111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112 113 /* get_task_state(): */ 114 #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ 115 TASK_UNINTERRUPTIBLE _ TASK_STOPPED \ 116 TASK_IRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED) 118		#define TASK_IDLE	(TASK_UNINTERRUPTIBLE TASK_NOLOAD)					
111 #define TASK_NORMAL (TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE) 112 113 /* get_task_state(): */ 114 #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ 115 TASK_UNINTERRUPTIBLE _ TASK_STOPPED \ 116 TASK_IRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED) 118								
112 113 /* get_task_state(): */ 114 #define TASK_REPORT 115 TASK_UNINTERRUPTIBLE \ 116 TASK_IRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED) 118								
113 /* get_task_state(): */ 114 #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ 115 TASK_UNINTERRUPTIBLE TASK_STOPPED \ 116 TASK_IRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED) 118		#define TASK_NORMAL	(TASK_INTERRUPTIBLE TASK_UNINTERRUPTIBLE)					
114 #define TASK_REPORT (TASK_RUNNING TASK_INTERRUPTIBLE \ 115 TASK_UNINTERRUPTIBLE TASK_STOPPED \ 116 TASK_TRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED) 118								
115 TASK_UNINTERRUPTIBLE TASK_STOPPED \ 116 TASK_TRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED) 118								
116 TASK_TRACED EXIT_DEAD EXIT_ZOMBIE \ 117 TASK_PARKED) 118		#detine TASK_REPORT						
117 TASK_PARKED) 118								
118								
			TASK_PARKED)					
		<pre>#define task_is_running(task)</pre>	(READ_ONCE((task)->state) == TASK_RUNNING)					-

Diagram of Process State

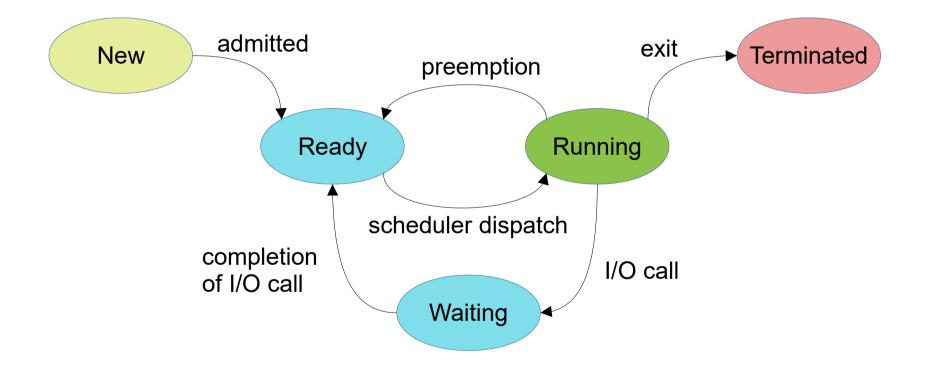
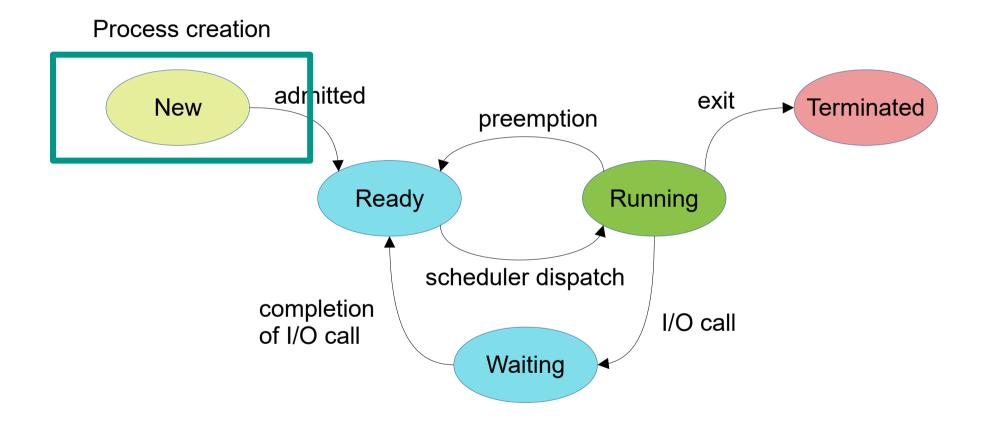


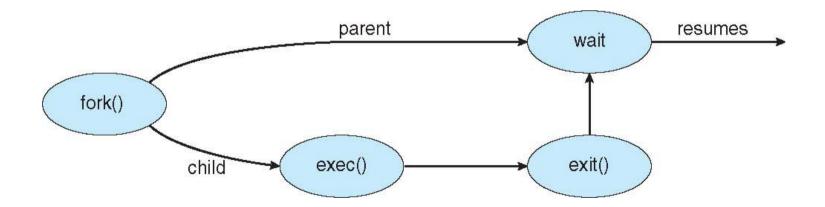
Diagram of Process State



Process Creation Same as before Parent Parent Child Copy of parent Before fork After fork

Unix example

- fork() system call creates new process
- *exec*() system call used after a fork() to replace the process' memory space with a new program



More on fork

- fork returns the process id of the child process
 - This is the only way for a process to know if it is the parent or child after a fork

```
int pid = 0;
pid = fork();
// for child: pid == 0, for parent: pid == <child
process id>
```

In Pintos fork is integrated in exec (not in normal Unix)

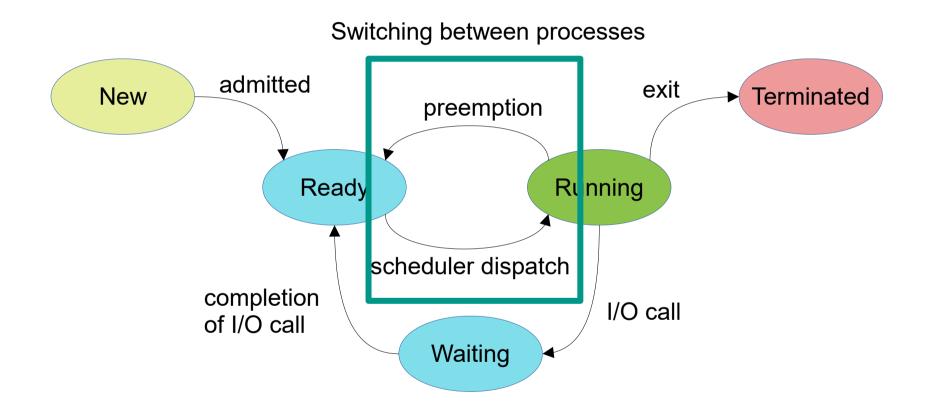
Poll question

```
int main(void) {
  int pid1 = 0;
  pid1 = fork();
  if (pid1 == 0) {
    printf("Hello\n");
  }
  int pid2 = fork();
  printf("Hello\n");
  sleep(1);
  return 0;
```

How many times will the program output the line "Hello"?

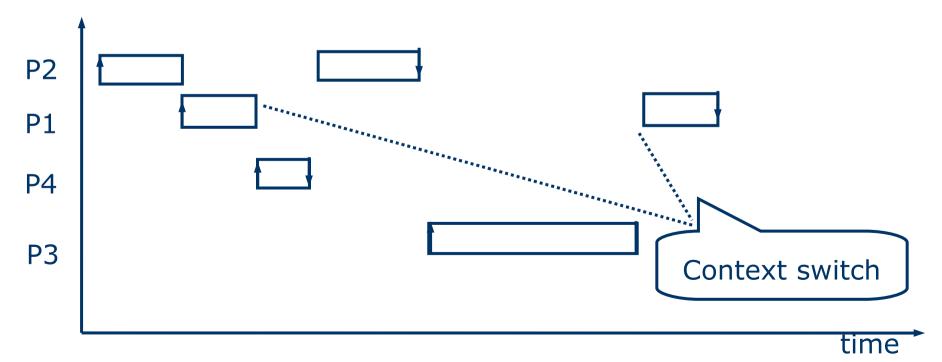
URL: https://www.menti.com Code: 8619 0811

Diagram of Process State



Context Switch

- Consider a program has several processes
 P1, ..., P4
- An execution of the concurrent program may look like:



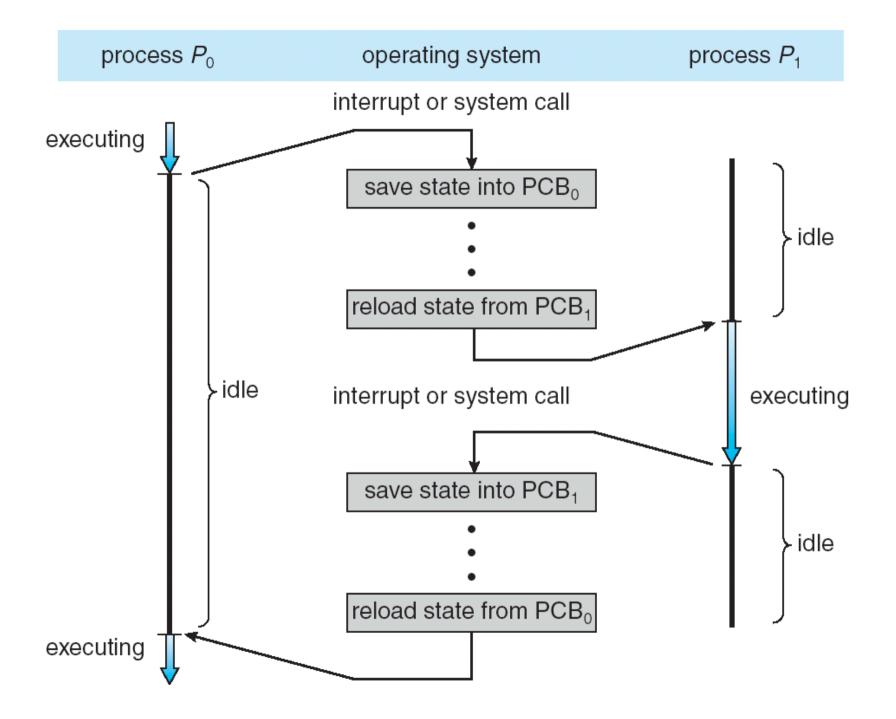
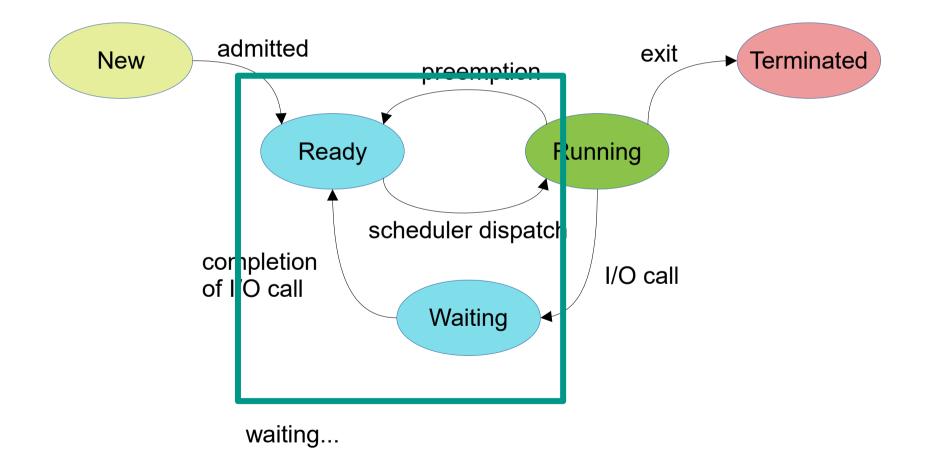
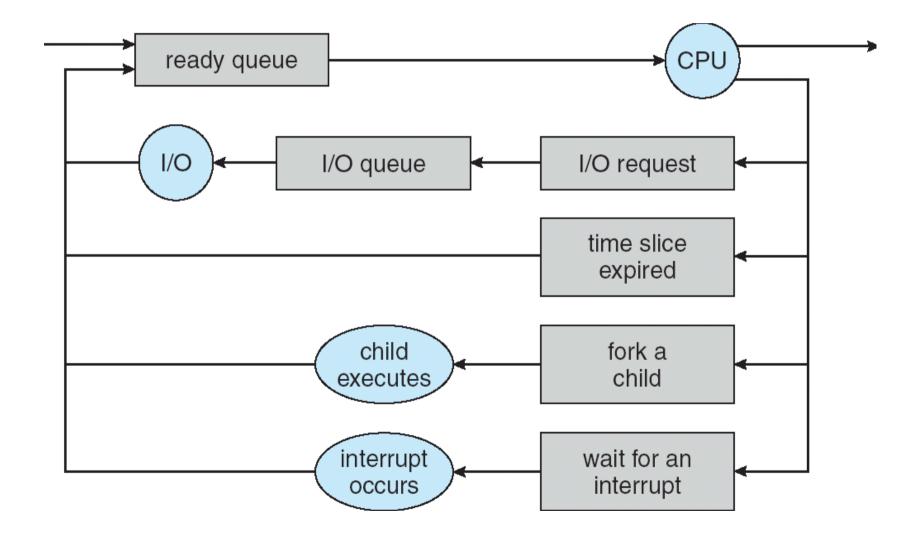


Diagram of Process State

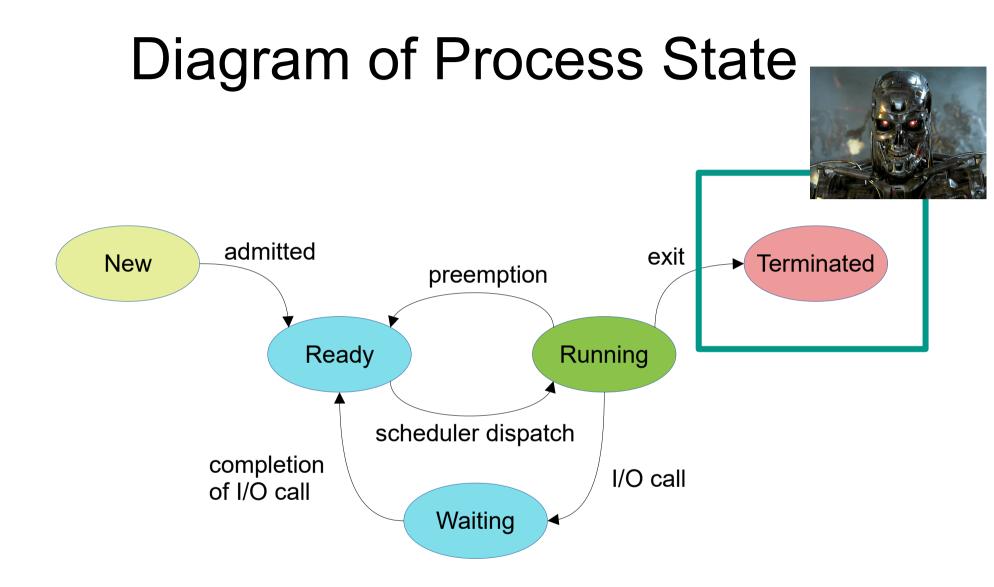


What are processes typically doing?

Process queues



For every reason to wait there is a queue. And then there is the ready queue.



Process termination

- Exit a process by calling exit(EXIT_SUCCESS)
 - (same as return 0)
- OS able to free resources
 - Take back memory, close open files, etc
 - All of it? No!

Parents must care for their children

Using processes to run a task in the background:

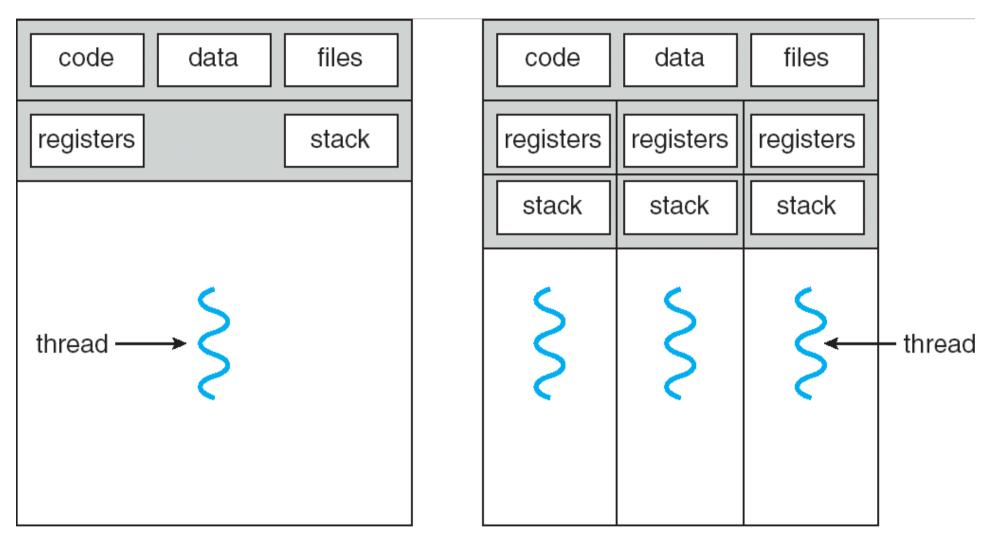
pid t pid = fork(); if (pid == 0) { // This is run by the child // Do something useful } else { // This is run by the parent // Continue with own stuff int status; wait(&status) // Wait until child is done

What happens if

- The parent does not call wait?
 - The finished child becomes a **zombie** process
- The parent terminates before the child?
 - The child becomes an **orphan** process (can be adopted by the init process)

Threads

Single- and Multithreaded Processes



multithreaded process

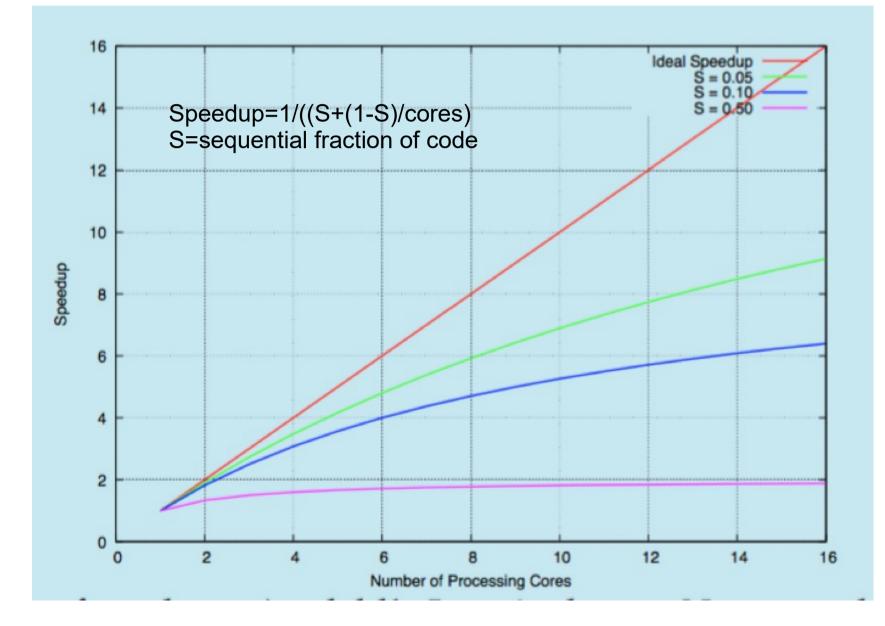
Benefits of Multithreading

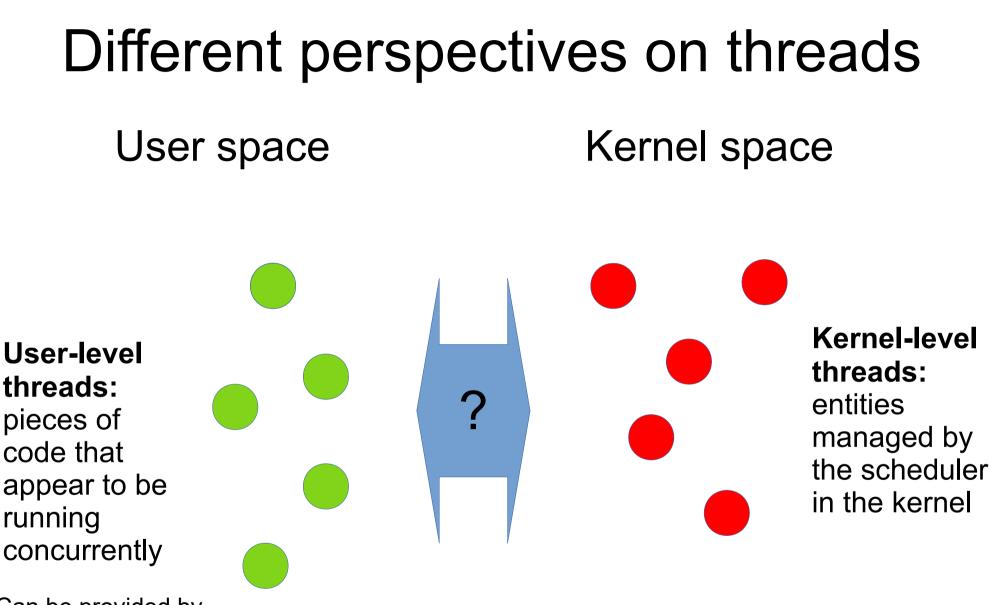
- Responsiveness
 - Interactive application can continue even when part of it is blocked
- Resource Sharing
 - Threads of a process share its memory by default.
- Economy
 - Light-weight
 - Creation, management, context switching for threads is much faster than for processes
 - E.g. Solaris: creation 30x, switching 5x faster
- Utilization of multicore architectures

Multi-core architectures

- Necessitated by the end of Moore's Law
 - We can no longer keep making chips smaller (and thereby faster)
- Problem: Amdahl's Law...

Multi-core architectures



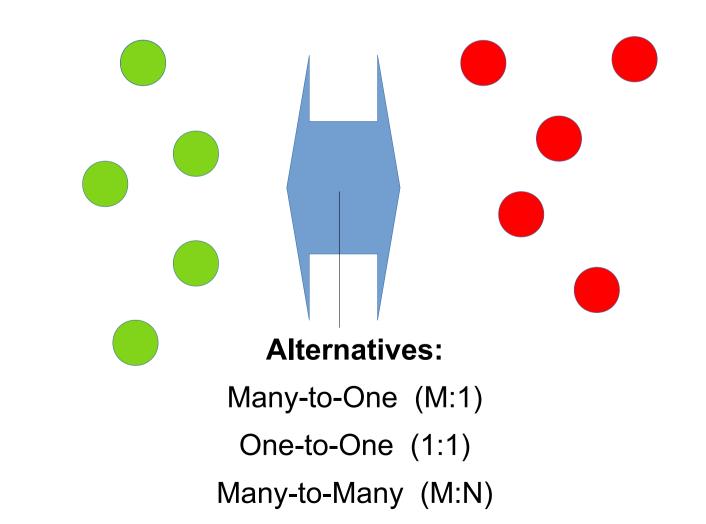


Can be provided by Runtime environment of programming language Special libraries Operating system Kernel-level threads ≠ Kernel threads

Mapping from user to kernel

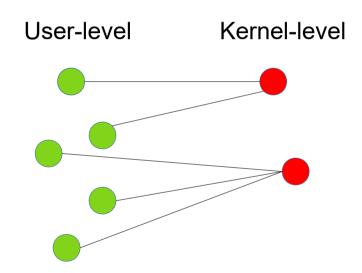
User-level threads

Kernel-level threads



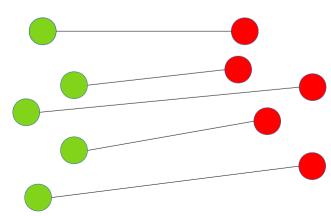
Many-to-One

- Many user-level threads mapped to single kernel thread
- Cow overhead, very portable
- Not scalable to multiprocessors, does not handle blocking calls well
 - Example:
 - GNU Portable Threads



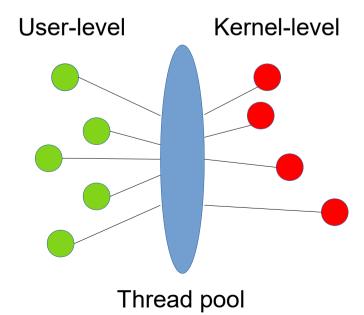
One-to-One

- Each user-level thread maps to one kernel thread
- more concurrency; scalable to multiprocessors
- Overhead of creating a kernel thread for each User-level Kernel-level
 Kernel-level
- Examples
 - Windows
 - Unix



Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the OS to create a sufficient number of kernel threads
- Abandoned by most OS:s
 - But used by threading libraries
 - Android, Java, ..



Java example

Implementing Runnable interface:

```
class Task implements Runnable
{
    public void run() {
        System.out.println("I am a thread.");
    }
}
```

Creating a thread:

```
Thread worker = new Thread(new Task());
worker.start();
```

Waiting on a thread:

```
try {
   worker.join();
}
catch (InterruptedException ie) { }
```

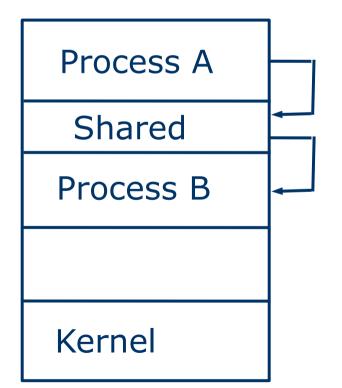
Implicit threading

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Examples
 - Thread Pools (Android, Grand Central Dispatch)
 - Fork-Join (cf. MapReduce)
 - OpenMP
 - Execution policies (C++)

Process interaction

Inter-process communication (IPC)

- Two modes:
 - Shared variables
 - Message passing



Process A	
Process B	
Kernel	

Message passing model

- Resembles a distributed system
- Benefits
 - Clean separation of data
 - Easy to distribute across multiple computers
 - Low risk of data corruption
- Drawbacks
 - Distributed algorithms are difficult to construct
 - Might result in bad performance

File systems (I)

File system consists of interface + implementation

Storing data

- Primary memory is volatile
 - need secondary storage for long-term storage
- A *disk* is essentially a linear sequence of numbered blocks
 - With 2 operations: write block *b*, read block *b*
 - Low level of abstraction

The file abstraction

- Provided by the OS
- Smallest allotment of secondary storage known to the user
 - Typically, contiguous logical address space
- Organized in a *directory* of files
- Has
 - Attributes (Name, id, size, ...)
 - API (operations on files and directories)

Meta data

- File attributes name, date of creation, protection info, ...
- Such information *about* files (i.e., meta-data) is kept in a directory structure, which is maintained on the disk.
- Stored in a File Control Block (FCB) data structure for each file

Open in Unix:

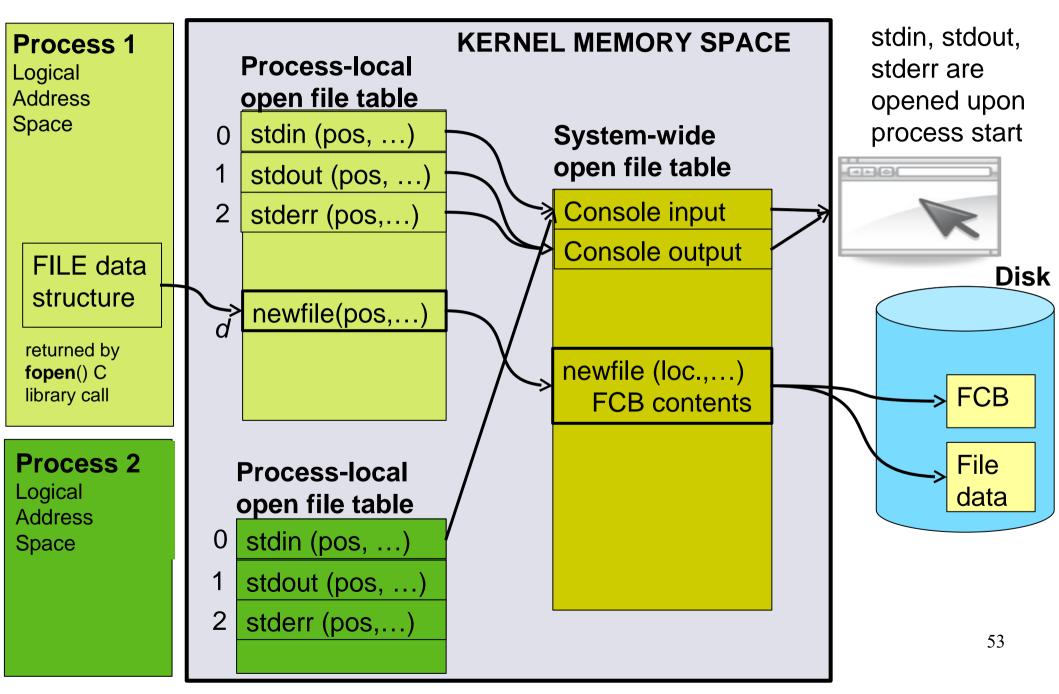
open ("filename", "mode")

returns a *file descriptor / handle* = index into a per-process \rightarrow

table of open files (part of PCB)

(or an error code)

File descriptors and open file tables

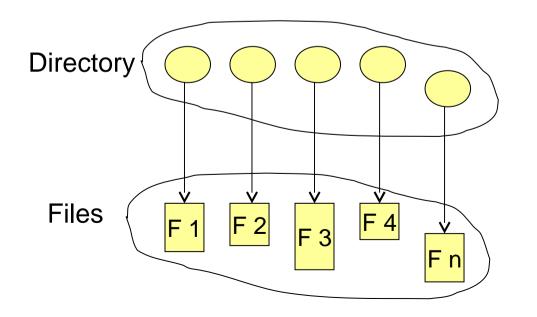


Storing open file data

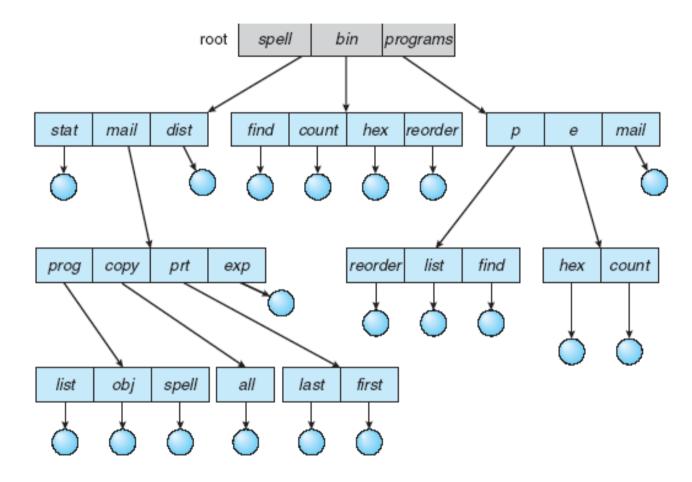
- Collected in a system-wide table of open files and process-local open file tables (part of PCB)
- Process-local open file table entries point to system-wide open file table entries
- Semantics of fork()?

Directory Structure

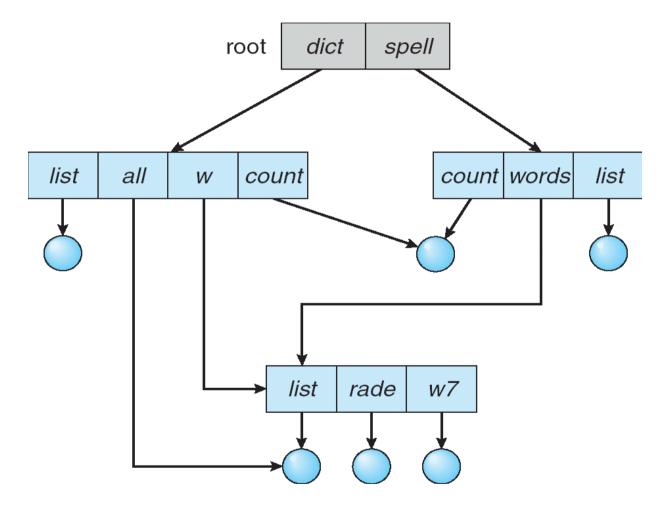
- Files in a system organised in directories
 - A collection of *nodes* containing information about all files
- Both the directory structure and the files reside on disk.



Tree-Structured Directories



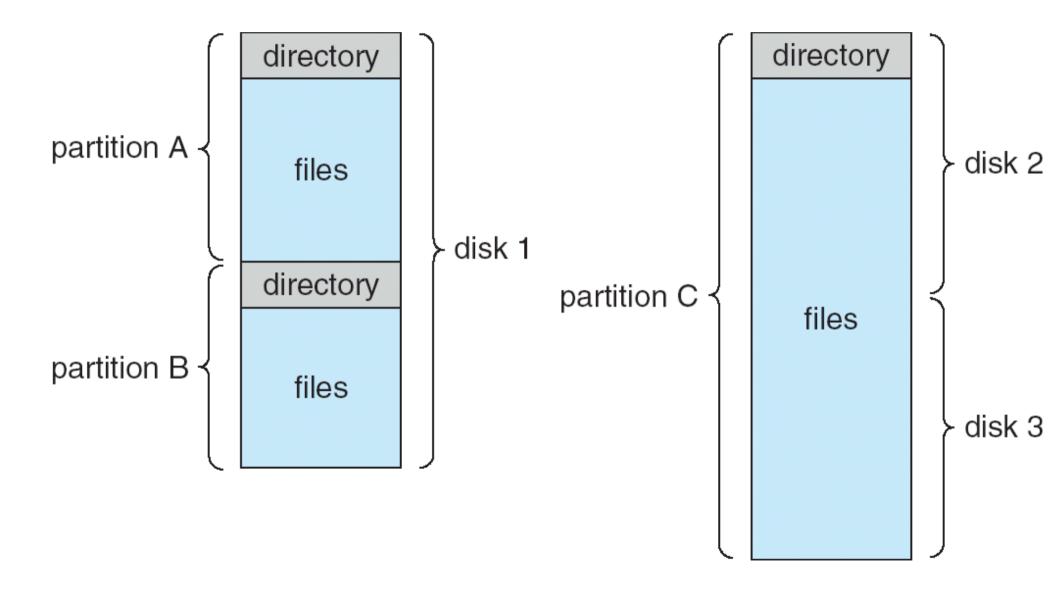
Acyclic-Graph Directories



Links

- Hard links
 - Direct pointer (block address) to a directory or file
 - Cannot span partition boundaries
 - Need be updated when file moves on disk
- Soft links (symbolic links, "shortcut", "alias")
 - files containing the actual (full) file name
 - still valid if file moves on disk
 - no longer valid if file name (or path) changes
 - not as efficient as hard links (one extra block read)

Examples of File-system Organization



File System Mounting

- A file system must be mounted before it can be accessed
- Mounting combines multiple file systems in one namespace
- An unmounted file system is mounted at a mount point
- In Windows, mount points are given names C:,
 D:, ...

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a protection scheme
- In order to have a protection scheme, the system should have
 - User IDs identify users, allowing permissions and protections to be peruser
 - Group IDs allow users to be in groups, permitting group access rights

Sharing across a network

- Distributed system
- Network File System (NFS) is a common distributed file-sharing method
- SMB (Windows shares) is another
- Protection is a challenge!

What's next

- Next lecture: Lecture 4
 - Scheduling processes and tasks (good to know for Lab3 - suspended timer_sleep impl.)
 - Reading: Ch. 5.1-5.5, 5.8
- Lecture 5: Synchronization, tools to manage data sharing between multiple processes